AD-A056 889 ILLINOIS UNIV AT URBANA-CHAMPAIGN COORDINATED SCIENCE LAB F/G 12/1
FINDING THE INTERSECTION OF TWO CONVEX POLYHEDRA.(U)
OCT 77 D E MULLER, F P PREPARATA DAAB07-72-C-0259
UNCLASSIFIED R-793 NL

| OF |
AD
A056889

END
DATE
FILMED
9-78
DDC

(11)

ACT-6 OCTOBER, 1977

COORDINATED SCIENCE LABORATORY

APPLIED COMPUTATION THEORY GROUP

FINDING THE INTERSECTION OF TWO CONVEX POLYHEDRA

D.E. MULLER and F.P. PREPARATA

REPORT R-793

UILU-ENG 77-2240

UNIVERSITY OF ILLINOIS – URBANA, ILLINOIS

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 FINDING THE INTERSECTION OF TWO CONVEX POLYHEDRA		5. TYPE OF REPORT & PERIOD COVERED 9 Technical Rep[redacted]t
		6. PERFORMING ORG. REPORT NUMBER R-793; UILU-ENG 77-2240
7. AUTHOR(s) 10 D. E. Muller [redacted] F. P. Preparata		8. CONTRACT OR GRANT NUMBER(s) MCS 76-17321 DAAB-07-72-C-0259
9. PERFORMING ORGANIZATION NAME AND ADDRESS Coordinated Science Laboratory University of Illinois at Urbana-Champaign Urbana, Illinois 61801		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Joint Services Electronics Program		12. REPORT DATE 11 Oct[redacted]77
		13. NUMBER OF PAGES 32
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) 12 38 p.		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

15 DAAB07-72-C-0259, NSF-MCS-76-17321

18. SUPPLEMENTARY NOTES

14 R-793, UILU-ENG-77-2240

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computational Complexity
Computational Geometry
Analysis of Algorithms
Polyhedra
Intersection of Polyhedra
Separating Plane
Linear Separability
Geometric Duality
Convex Hull

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Given two convex polyhedra in three-dimensional space, we develop an algorithm to (i) test whether their intersection is empty, and (ii) if so to find a separating plane, while (iii) if not to find a point in the intersection and explicitly construct their intersection polyhedron. The algorithm runs in time O(nlogn), where n is the sum of the numbers of vertices of the two polyhedra. The part of the algorithm concerned with (iii) (constructing the intersection) is based upon the fact that if a point in the intersection is known, then the entire intersection is obtained from the convex hull of suitable

20. ABSTRACT (continued)

geometric duals of the two polyhedra taken with respect to this point.

LEVEL II

11

UILU-ENG 77-2240

FINDING THE INTERSECTION OF TWO CONVEX POLYHEDRA

by

D. E. Muller and F. P. Preparata

This work was supported in part by the National Science Foundation under Grant MCS76-17321 and in part by the Joint Services Electronics Program (U.S. Army, U.S. Navy and U.S. Air Force) under Contract DAAB-07-72-C-0259.

FINDING THE INTERSECTION OF TWO CONVEX POLYHEDRA

D. E. Muller* and F. P. Preparata†
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign

Abstract

Given two convex polyhedra in three-dimensional space, we develop an algorithm to (i) test whether their intersection is empty, and (ii) if so to find a separating plane, while (iii) if not to find a point in the intersection and explicitly construct their intersection polyhedron. The algorithm runs in time O(nlogn), where n is the sum of the numbers of vertices of the two polyhedra. The part of the algorithm concerned with (iii) (constructing the intersection) is based upon the fact that if a point in the intersection is known, then the entire intersection is obtained from the convex hull of suitable geometric duals of the two polyhedra taken with respect to this point.

*Also, Departments of Mathematics and of Computer Science.

†Also, Departments of Electrical Engineering and of Computer Science

This work was supported in part by the National Science Foundation under Grant MCS76-17321 and in part by the Joint Services Electronics Program under Contract DAAB-07-72-C-0259.

1. Introduction

Finding the intersection of two convex polyhedra in three-dimensional space is a classical problem in computational geometry [1]. A simple but time-consuming solution to this problem is known, and it is so trivial that it is not even worth a literature reference. It goes as follows. Let $\mathcal{A}$ and $\mathcal{B}$ be two convex polyhedra, test each face of $\mathcal{A}$ against each face of $\mathcal{B}$ to see if they intersect; if no intersection is found, then the intersection of the two polyhedra is empty, otherwise it can be simply constructed. It is clear that such an algorithm could use $O(n^2)$ operations if n is the sum of the numbers of vertices of $\mathcal{A}$ and $\mathcal{B}$.

The search for a more efficient procedure has in the past met with no success. Several facts, however, suggested that a more efficient method should exist: first, it is well-known that two polygons in the plane can be intersected in time linear in the sum of their numbers of vertices [2]; second, several analogies exist between the plane and the space, i.e., convex hulls of n-point sets [3] and maxima of sets of n vectors [4] can be found in time O(nlogn) in both two and three dimensions. In spite of this, no generalization of the polygon intersection algorithm has been found.

In special cases, however, better than quadratic time methods have been known for some time. Specifically, after the development of the Lee-Preparata algorithm for locating a point in a planar subdivision [5], it was realized(1) that the intersection of two polyhedra can be found in time $O(n\log^2 n)$ if a vertex of one polyhedron lies inside the other polyhedron.

(1) Private communications between M. I. Shamos and F. P. Preparata, May 1976. See also [1] p. 160.

Alternately (see [1], p. 162), Shamos conjectured that polyhedron intersection could be obtained as a merge step of a divide-and-conquer algorithm for the intersection of half-spaces.

In this paper we present an algorithm for solving the problem of intersecting two polyhedra in time O(nlogn). The algorithm tests whether the intersection is empty and, if not, explicitly constructs it. The approach is based on the fact that, if a point p in the intersection is known, the intersection can be obtained through geometric dualization. Specifically, the two polyhedra are both transformed into their geometric duals with respect to p, and the convex hull of the dual,which can be found in time O(nlogn), is the dual of the intersection polyhedron (Section 3). When such a special point p is not available,by deploying known techniques in time O(nlogn) we can test whether the intersection is nonempty and, if so, obtain a point in the intersection (Section 4). The algorithm requires that each polyhedron be represented by a very versatile data structure called the doubly connected edge list, which can be obtained from a more conventional representation in time linear in the number of vertices (Section 2).

2. Derivation of a Doubly Connected Edge List for a Planar Graph

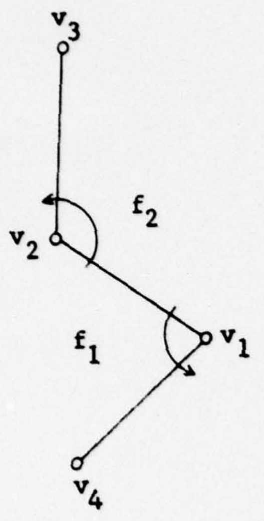
Let $V = \{v_1,\ldots,v_n\}$ and $E = \{e_1,\ldots,e_m\}$ be the sets of vertices and edges respectively, of a planar graph embedded in the plane without crossing edges. We assume that (V,E) be represented as follows. To vertex $v_j \in V$ there corresponds cell H[j] of an array H[1:n], which contains a pointer to the first term of the cyclic list of the edges incident on v_j, arranged in the order in which they appear as one proceeds counterclockwise around v_j. The latter lists are realized by means of two arrays VERTEX[1:Rm] and NEXT[1:2m] so that (VERTEX[i],NEXT[i]) is the format of the list nodes. This representation of the graph (V,E) is precisely the one obtained by the algorithm of Preparata and Hong [3] which constructs the convex hull of a set of points in three dimensions: indeed the surface of a convex polyhedron is topologically a planar graph. We shall call this collection of lists the vertex-to-edge representation of a planar graph.

Although the vertex-to-edge list is one of the most commonly used representations for a planar graph, it has the disadvantage that the dual graph, i.e., the set of faces of the original graph, is not readily available. For this one would have to develop the face-to-edge representation of the original graph in which each face refers to a cyclically ordered list of edges which enclose it.

A more convenient representation for this purpose is one which we shall call the doubly connected edge list (DCEL), from which we can obtain equally easily information either about the edges incident on a vertex or the edges enclosing a face. We will now describe the DCEL and give in an appendix the algorithm by which to obtain it from the more conventional vertex-to-edge representation of the graph in time proportional to the number n of vertices.

The main component of the DCEL of a planar graph (V,E) is the <u>edge node</u>. There is a one-to-one correspondence between edges and edge nodes, i.e., each edge is represented only once, whereas in the vertex-to-edge list each edge appeared twice. An edge node consists of <u>four</u> information fields V1, V2, F1, and F2 and two pointer fields P1 and P2: therefore the corresponding data structure is easily implemented with six arrays with the same names, each consisting of m cells. The meanings of these fields are as follows. The field V1 contains the name of the vertex which is the <u>origin</u> of the edge , whereas V2 contains the <u>terminus</u>; in this manner, the edge receives a conventional orientation. The field F1 and F2 contain the names of the faces which lie respectively on the left and on the right of the edge oriented from V1 to V2. The pointer P1 (P2) points to the edge node containing the first edge encountered after V1(V2) when one proceeds counterclockwise around V1 (V2). Names of faces and vertices may be taken as integers. As an example, a fragment of a graph and the corresponding fragment of the DCEL is shown in figure 1.

It is now easy to see how the edges incident on a given vertex or the edges enclosing a given face can be obtained from the DCEL. If the graph has n vertices and f faces, we can assume we have two arrays HV[1:n] and HF[1:f] of headers of the vertex and face lists: these arrays can be filled by a scan of arrays V1 and F1 in time O(n). The following straightforward procedure, VERTEX(j), obtains the sequence of edges incident on v, as a sequence of addresses stored in an array A.



	V1	V2	F1	F2	P1	P2
1						
2						
⋮						
a_1	1	2	1	2	a_2	a_3
a_2	4	1	1			
a_3	2	3		2		

Figure 1. Illustration of the DCEL.

```
                    VERTEX(j)
1. begin  $A[1] \leftarrow a_0 \leftarrow a \leftarrow HV[j],\ i \leftarrow 2$
2.        If $V1[a] = j$ then $a \leftarrow P1[a]$ else $a \leftarrow P2[a]$
3.        While $a \neq a_0$ do
4.          begin $A[i] \leftarrow a$
5.                If $V1[a] = j$ then $a \leftarrow P1[a]$ else $a \leftarrow P2[a]$
6.                $i \leftarrow i+1$
            end
   end
```

Clearly VERTEX(j) runs in time proportional to the number of edges incident on v_j. Analogously, we can develop a procedure, FACE(j), which obtains the sequence of edges enclosing f_j, by replacing HV and V1 with HF and F1, respectively, in the above procedure VERTEX (j). Notice that the procedure

VERTEX traces the edges counterclockwise about a vertex while FACE traces them clockwise about a face.

In an appendix to this paper we shall show that the DCEL of a planar graph can be obtained from its vertex-to-edge list in time linear in the number of vertices.

3. Finding the Intersection of Two Polyhedra when a Point in the Intersection is Known.

Assume that the two convex polyhedra $\mathcal{A}$ and $\mathcal{B}$ in three-dimensional Euclidean space are represented by DCEL's. They are taken as having vertex sets $V_{\mathcal{A}}$ and $V_{\mathcal{B}}$ and face sets $F_{\mathcal{A}}$ and $F_{\mathcal{B}}$ respectively. With each vertex $v \in V_{\mathcal{A}} \cup V_{\mathcal{B}}$ we associate its three Cartesian coordinates $x_1(v)$, $x_2(v)$, $x_3(v)$. With each face $f \in F_{\mathcal{A}} \cup F_{\mathcal{B}}$ we associate a half-space bounded by its plane and containing the corresponding polyhedron. The half-space of f is described by the inequality

$$n_1(f)x_1 + n_2(f)x_2 + n_3(f)x_3 + d(f) \geq 0 \qquad (1)$$

where $n_1(f)$, $n_2(f)$, $n_3(f)$ and $d(f)$ are four parameters characteristic of f, normalized so that d(f) is either 1, 0, or -1. Our objective is to obtain a DCEL for the intersection of $\mathcal{A}$ and $\mathcal{B}$ along with the coordinates of its vertices and the parameters for its faces.

In this portion of the analysis it is assumed that a point in the intersection of $\mathcal{A}$ and $\mathcal{B}$ is known: the problem of finding such a point will be discussed later. Without loss of generality, the point in the intersection will be taken as the origin, since if it is not the origin a simple translation of coordinates will make it the origin. We begin by assuming that the origin is actually in the interior of each of the two polyhedra and hence in the interior of their intersection. In this case each face $f \in F_{\mathcal{A}} \cup F_{\mathcal{B}}$ represents a half-space of the form $n_1(f)x_1 + n_2(f)x_2 + n_3(f)x_3 + 1 \geq 0$. In other words, the constant $d(f) = 1$ for all faces f.

Now, for each of the polyhedra $\mathcal{A}$ or $\mathcal{B}$ there is a corresponding polyhedron $\mathcal{A}^{(D)}$ or $\mathcal{B}^{(D)}$, respectively, which we shall call its dual. The dual $\mathcal{A}^{(D)}$ of $\mathcal{A}$ is obtained by reinterpreting the coefficients $n_1(f)$, $n_2(f)$, $n_3(f)$ of each face $f \in F_{\mathcal{A}}$ as the coordinates of a corresponding vertex v_f of $\mathcal{A}^{(D)}$. Conversely, the coordinates $x_1(v)$, $x_2(v)$, $x_3(v)$ of each vertex of $V_{\mathcal{A}}$ are reinterpreted as the coefficients of a corresponding face f_v of $\mathcal{A}^{(D)}$. This transformation may be regarded as a conventional dualization about the unit sphere with center at the origin, where points at distance ℓ from the origin are transformed into planes at distance $1/\ell$ from the origin and vice versa ([7], p.233). A similar procedure allows us to form the dual of $\mathcal{B}$.

We note that this dualization procedure is only possible because the origin is in the interior of the polyhedron. In this case, the dual is also a convex polyhedron containing the origin. If the origin is not in the interior, then some of the inequalities (1) would require $d(f) = 0$ or -1, and we have not defined dual points for such half-spaces.

Let $V_{\mathcal{A}}^{(D)}$ and $V_{\mathcal{B}}^{(D)}$ be the vertex sets of $\mathcal{A}^{(D)}$ and $\mathcal{B}^{(D)}$ respectively. It is easily seen that the convex hull of the union of $\mathcal{A}^{(D)}$ and $\mathcal{B}^{(D)}$ is the dual of the intersection of $\mathcal{A}$ and $\mathcal{B}$. Hence, in order to find the intersection of $\mathcal{A}$ and $\mathcal{B}$ we may simply use the algorithm of Preparata and Hong [1] to find the convex hull of the set of vertices $V_{\mathcal{A}}^{(D)} \cup B_{\mathcal{A}}^{(D)}$ in time $O(n\log n)$, and upon taking the dual of the result we obtain the desired polyhedron.

Now let us assume that the given point in the intersection of $\mathcal{A}$ and $\mathcal{B}$, i.e., the origin, is not in the interior of their intersection. Then certain faces $f \in F_{\mathcal{A}} \cup F_{\mathcal{B}}$ have $d(f) = 0$. In fact, these are exactly the faces which pass through the origin. Let F' be the set of such faces. To each face $f' \in F'$ there is a corresponding inequality of the form

$$n_1(f')x_1 + n_2(f')x_2 + n_3(f')x_3 \geq 0, \qquad (2)$$

obtained from (1) by replacing $d(f')$ by 0. A point x in the interior of $\mathcal{A} \cap \mathcal{B}$ must strictly satisfy all inequalites of type (1) with $f \in F_{\mathcal{A}} \cup F_{\mathcal{B}}$, that is, none can be an equality. Such a point x exists if and only if all inequalities of type (2) with $f' \in F'$ can be satisfied strictly by some point. To determine whether there is such a point we first fix $x_3 = 1$ and write the strict form of (2) as

$$n_1(f')x_1 + n_2(f')x_2 > - n_3(f'). \qquad (2')$$

Here, by normalizing the coefficients, we can take $- n_3(f')$ as either 1, 0, or -1.

The question of whether or not (2') can be satisfied for every $f' \in F'$ is a two-dimensional problem of the type we are solving here for the three-dimensional case. The inequalities of (2') collectively represent a two-dimensional convex set which can be found ([1], p. 158) in time O(nlogn). Actually, a faster computation of this convex set is possible: the inequalities of (2') can be partitioned into two sets, depending upon which of the two polyhedra they pertain to; each such set corresponds to a polygon in the plane $x_3 = 1$, and the desired convex set is the intersection of the two polygons. It is known that finding these polygons and their intersection runs in time at most proportional to n [1].

If no solution is found in the above case, the case $x_3 = -1$ must also be tried. This problem is similar to the previous one except that $-n_3(f')$ is replaced by $n_3(f')$ in all the inequalities (2').

Let us suppose that we have been able to strictly satisfy all inequalities of the type (2) with $x_3 = 1$ or -1 using the above method. Clearly, they will remain satisfied if the vector x is multipled by a positive scalar. To strictly satisfy all the remaining inequalities in (1) whose right hand sides are all 1, we simply choose such a scalar which makes all the left hand sides less than 1. The resulting point is in the interior of $\mathcal{A} \cap \mathcal{B}$.

If it is impossible to strictly satisfy all the inequalities of (2), then any one which cannot be strictly satisfied, say

$$n_1(f'')x_1 + n_2(f'')x_2 + n_3(f'')x_3 = 0$$

represents a plane through the origin which contains the intersection of $\mathcal{A} \cap \mathcal{B}$. Thus, the intersection of $\mathcal{A}$ and $\mathcal{B}$ may be found entirely within this plane. This problem is analogous to the one discussed before and can be solved, as we saw, in time O(n).

4. Finding a Point in the Intersection of Two Polyhedra

In the preceding section we have shown that the intersection of two convex polyhedra can be obtained when a point in the intersection is known. Thus, if the intersection is nonempty, all that is needed is to find one such point. The objective of this section is the implementation of this task.

Given a convex polyhedron $\mathcal{A}$, a plane is called a plane of support of $\mathcal{A}$ if it has at least one point in common with $\mathcal{A}$ and all interior points of $\mathcal{A}$ lie on one side of the plane. Hereafter we shall only consider planes of support parallel to the x_3-axis and briefly refer to them as vertical. The intersection of $\mathcal{A}$ with its vertical planes of support is, in general, an annular region $R(\mathcal{A})$ of the surface $\mathcal{A}$ which, in the absence of degeneracies, reduces to a cycle of edges. The projection of $R(\mathcal{A})$ on the (x_1,x_2) plane is a convex polygon $\mathcal{A}^*$ (figure 2), which is the convex hull of the projections of the points of $\mathcal{A}$ on this plane.

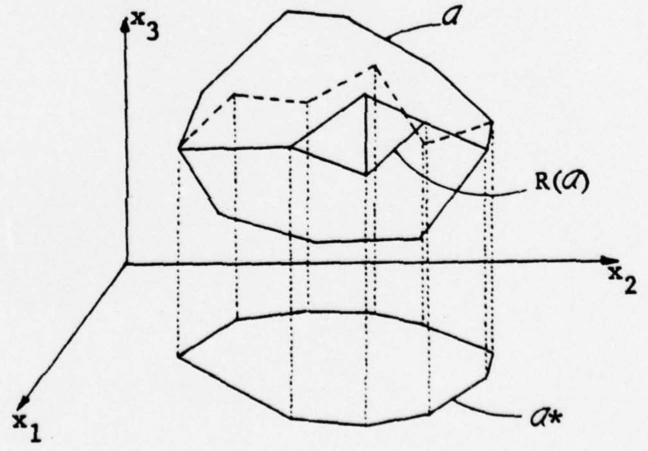


Figure 2. A convex polyhedron $\mathcal{A}$, the annular region $R(\mathcal{A})$ and the projection polygon $\mathcal{A}^*$.

The region $R(\mathcal{A})$ is easily obtained from the DCEL description of $\mathcal{A}$ as follows. For any face f_i of $\mathcal{A}$, the <u>normal</u> to f_i is the vector $(n_1(f_i),n_2(f_i),n_3(f_i))$. It is perpendicular to f_i and points toward the interior of $\mathcal{A}$. Given any edge e of $\mathcal{A}$, let f_i and f_j be its adjacent faces. Then $e \in R(\mathcal{A})$ if and only if

$$n_3(f_i)\cdot n_3(f_j) \leq 0. \tag{3}$$

Therefore, we begin by scanning the edge set of $\mathcal{A}$ until we find an edge e which belongs to $R(\mathcal{A})$, by verifying condition (3). At this point, we select one of the two vertices of e, call it v. Among the edges incident on v there are either one or two new edges, different from e, which belong to $R(\mathcal{A})$ and can be easily found by applying the VERTEX procedure described earlier, to the DCEL. Thus we can advance in the construction of $R(\mathcal{A})$, which will be completed upon re-encountering the initial edge e. Once $R(\mathcal{A})$ has been computed, $\mathcal{A}^*$ is trivially obtained. All of these operations can be carried out in time proportional to the number $|V_{\mathcal{A}}|$ of the vertices of $\mathcal{A}$. Thus we have the first steps of the algorithm, given polyhedra $\mathcal{A}$ and $\mathcal{B}$ with $|V_{\mathcal{A}}| + |V_{\mathcal{B}}| = n$:

<u>Step 1</u>. Find $\mathcal{A}^*$ and $\mathcal{B}^*$. (This step runs in time $O(n)$.)

<u>Step 2</u>. Using Shamos-Hoey's polygon intersection algorithm [1], find the intersection of $\mathcal{A}^*$ and $\mathcal{B}^*$. If the intersection is empty, halt, for $\mathcal{A} \cap \mathcal{B}$ is also empty. Else let p^* be a point in the intersection of $\mathcal{A}^*$ and $\mathcal{B}^*$. (This step runs in time $O(n)$, according to [1].)

Under the projection of $\mathcal{A}$ to $\mathcal{A}^*$, $p^* = (x_1(p^*),x_2(p^*))$ is in general the image of a vertical segment of $\mathcal{A}$ which reduces to a single point in some cases. In any case, the preimage of p^* in $\mathcal{A}$ is easily found in time $O(n)$ as follows. For each face $f \in F_{\mathcal{A}}$ we determine the x_3-coordinate of the point on the corresponding plane which projects to p^*; specifically, this x_3-coordinate is

$$\alpha(f) = -(n_1(f)x_1(p^*) + n_2(f)x_2(p^*) + d(f))/n_3(f).$$

Let $\alpha' = \min_{n_3(f) < 0} \alpha(f)$ and $\alpha'' = \max_{n_3(f) > 0} \alpha(f)$. Then α' and α'', with $\alpha' \geq \alpha''$, are the x_3-coordinates of the extremes of the segment which is the preimage of p^* in $\mathcal{A}$; we similarly define β' and β'', with $\beta' \geq \beta''$, for the analogous segment in $\mathcal{B}$. If the two segments overlap, then any point in their common portion also belongs to the nonempty intersection of $\mathcal{A}$ and $\mathcal{B}$. Otherwise assume, without loss of generality, that $\alpha'' > \beta'$. Then we define the <u>near-sides</u> of $\mathcal{A}$ and $\mathcal{B}$ to be the sets of faces $\{f_j | f_j$ is a face of $\mathcal{A}$, $n_3(f_j) < 0\}$ and $\{g_i | g_i$ is a face of $\mathcal{B}, n_3(g_i) > 0\}$, respectively. Clearly both near-sides are obtained in time $O(|V_{\mathcal{A}}| + |V_{\mathcal{B}}|) = O(n)$ by traversing, in a straightforward manner, the DCEL descriptions of $\mathcal{A}$ and $\mathcal{B}$. By projecting the near-sides of $\mathcal{A}$ and $\mathcal{B}$ on the (x_1,x_2)-plane we obtain two planar straight-line graphs (PSLG's) $G_{\mathcal{A}}$ and $G_{\mathcal{B}}$, with respective vertex sets $V'_{\mathcal{A}}$ and $V'_{\mathcal{B}}$. Thus we have:

<u>Step 3</u>. If the pre-images of p^* in $\mathcal{A}$ and $\mathcal{B}$ under an x_3-projection have a common intersection, then halt, for any point in this intersection is internal to $\mathcal{A} \cap \mathcal{B}$. Otherwise obtain $G_{\mathcal{A}}$ and $G_{\mathcal{B}}$. (The pre-images of p^* are found in constant time; $G_{\mathcal{A}}$ and $G_{\mathcal{B}}$ are found in time $O(n)$.)

Let $\mathcal{D}$ be the closed domain contained in the intersection of $\mathcal{A}^*$ and $\mathcal{B}^*$. For each point $u \in \mathcal{D}$ it is convenient to define the function $\delta(u)$, called

x_3-distance, as follows. If $\alpha(u)$ and $\beta(u)$ are the x_3-coordinates of the points on the faces of the near-sides of $\mathcal{A}$ and $\mathcal{B}$, respectively, which both project to u, then

$$\delta(u) = \alpha(u) - \beta(u) .$$

Let us now analyze the function $\delta(u)$ defined on $\mathcal{D}$. Imagine superimposing $G_{\mathcal{A}}$ and $G_{\mathcal{B}}$ to create a new vertex, conveniently called a pseudo-vertex, at the intersection of each edge of $G_{\mathcal{A}}$ with an edge of $G_{\mathcal{B}}$. Denoting by V^* the set of pseudo-vertices thus obtained, we can define a new PSLG G^* with vertex set $V'_{\mathcal{A}} \cup V'_{\mathcal{B}} \cup V^*$. The vertices of $V'_{\mathcal{A}}$ and $V'_{\mathcal{B}}$ will be called true vertices. Thus the domain $\mathcal{D}$ is subdivided into regions by G^*. Notice that inside any region of $G_{\mathcal{A}}$ the function $\alpha(v)$ is linear in the (x_1,x_2) coordinates at v; similarly, for the function $\beta(v)$ inside any region of $G_{\mathcal{B}}$. Thus in any region induced by G^* in $\mathcal{D}$, the function $\delta(v)=\alpha(v) - \beta(v)$ is linear in $x_1(v)$ and $x_2(v)$. Moreover, $\alpha(v)$ is convex-downward and $\beta(v)$ is convex-upward; it follows that $\delta(v)$ is a convex-downward function. We conclude that the minimum of δ occurs at a vertex of G^*. Notice that $|V^*|$, and hence $|V'_{\mathcal{A}} \cup V'_{\mathcal{B}} \cup V^*|$, could be $O(n^2)$: in fact, it is not hard to construct two planar graphs, each with ν vertices, so that, when superimposed, $(\nu - 1)^2$ intersections of edges are obtained.

Since, by hypothesis, $\alpha(p^*) = \alpha''$ and $\beta(p^*) = \beta'$, we conclude that $\delta(p^*) = \alpha'' - \beta' > 0$. It follows that the intersection of $\mathcal{A}$ and $\mathcal{B}$ is nonempty if and only if, for some $v \in \mathcal{D}$, $\delta(v) \leq 0$. Therefore, either we find one such point, or show that $\min_{v \in \mathcal{D}} \delta(v) > 0$.

To this end, we begin by evaluating δ at true vertices of G^* in $\mathcal{D}$. This is easily done if, for each $a \in V'_{\mathcal{A}}$ we determine the region $r(a)$ of $G_{\mathcal{B}}$ to which a belongs. If $r(a)$ is not the infinite region of the plane in the subdivision induced by $G_{\mathcal{B}}$, then $r(a)$ corresponds to a unique face of the polyhedron $\mathcal{B}$, and $\delta(a)$ is easily computed. Similarly, we can compute $\delta(b)$ for each $b \in V'_{\mathcal{B}}$ in $\mathcal{D}$. The determination of $r(a)$ has been called the location of point a in the planar subdivision induced by $G_{\mathcal{B}}$ [5] and could be done one point at a time. However, a faster algorithm has been recently developed [6] for collectively locating all the points of a set. According to this technique the members of $V'_{\mathcal{A}}$ can all be located in regions of the planar subdivision induced by $G_{\mathcal{B}}$ in time $O((c|V'_{\mathcal{A}}|+|V'_{\mathcal{B}}|)\log|V'_{\mathcal{B}}|)$, for some constant c, and, reciprocally, the members of $V'_{\mathcal{B}}$ can all be located in regions of the planar subdivision induced by $G_{\mathcal{A}}$ in time $O((|V'_{\mathcal{A}}|+c|V'_{\mathcal{B}}|)\log|V'_{\mathcal{A}}|)$. Therefore in total time $O((|V'_{\mathcal{A}}|+|V'_{\mathcal{B}}|)\log|V'_{\mathcal{A}}|\cdot|V'_{\mathcal{B}}|) = O((|V_{\mathcal{A}}|+|V_{\mathcal{B}}|)\log(|V_{\mathcal{A}}|\cdot|V_{\mathcal{B}}|)) = O(n\log n)$ all the true vertices can be located. Once a vertex, say,of $V'_{\mathcal{A}}$ has been located in $G_{\mathcal{B}}$, the x_3-coordinates of its preimages in $\mathcal{A}$ and $\mathcal{B}$ are obtained in constant time. This is summarized as follows:

Step 4. Locate each true vertex of $G_{\mathcal{A}}$ in the planar subdivision induced by $G_{\mathcal{B}}$ and vice versa. (This can be done in time $O(n\log n)$ using the algorithm of [6].) If there are no true vertices in $\mathcal{D}$ go to Step 7. Else evaluate δ at each true vertex of G^*. (This can be done in additional time $O(n)$.)

Suppose at first that there are true vertices in $\mathcal{D}$, and assume that for some true vertex v (say, $v \in V'_{\mathcal{A}}$) we have $\delta(v) \leq 0$. The vertical line through v intercepts the near-side of $\mathcal{A}$ in a vertex a and the near-side of $\mathcal{B}$ in a point b, and obviously $\alpha(v) = x_3(a) \leq x_3(b) = \beta(v)$. Thus, we have a point p^* for which $\alpha(p^*) > \beta(p^*)$ and a point v for which

$\alpha(v) \leq \beta(v)$. Consider now the plane, parallel to the x_3-axis and containing the points p* and v. The intersection of this plane with the two polyhedra $\mathcal{A}$ and $\mathcal{B}$ is shown in figure 3 (where the points p' and p" have been defined). By convexity, the segments $\overline{ap'}$ and $\overline{bp''}$ are entirely contained in $\mathcal{A}$ and $\mathcal{B}$ respectively, and so their point of intersection q belongs to the intersection of $\mathcal{A}$ and $\mathcal{B}$. The coordinates of q are thus obtained by straightforward calculations.

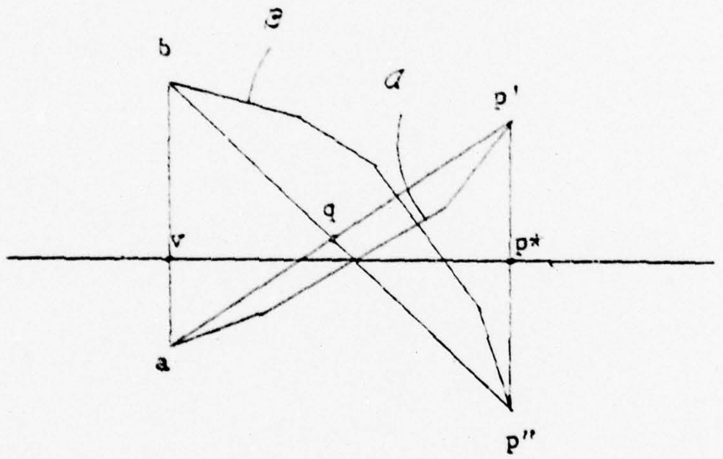


Figure 3. Finding a point in the intersection when $\delta(p^*) > 0$ and $\delta(v) \leq 0$.

Assume next that $\delta(v) > 0$ for all true vertices $v \in V'_{\mathcal{A}} \cup V'_{\mathcal{B}}$, and let v* be a true vertex such that $\delta(v^*) = \min \{\delta(v) | v \in V'_{\mathcal{A}} \cup V'_{\mathcal{B}}\}$. We cyclically test each of the edges of $\mathcal{A}$ incident upon v* to determine whether the function δ decreases as one moves along the edge away from v*. If it fails to decrease for all edges incident upon v*, then v* is an absolute minimum of the function δ. Since $\delta(v^*) > 0$, the polyhedra $\mathcal{A}$ and $\mathcal{B}$ do not intersect.

Step 5. Obtain v^*. If v^* is an absolute minimum and $\delta(v^*) > 0$, halt, for $\mathcal{A} \cap \mathcal{B} = \phi$; if $\delta(v^*) > 0$ but v^* is not an absolute minimum, go to Step 6; if $\delta(v^*) \leq 0$, then there is a point $q \in \mathcal{A} \cap \mathcal{B}$, obtained as the intersection of the diagonals of the trapezoid formed by the x_3-projection pre-images of p^* and v^* in $\mathcal{A}$ and $\mathcal{B}$. (All of this work can be done in time $O(n)$.)

The remaining case is when $\delta(v^*) > 0$ but v^* is not an absolute minimum. Then δ decreases as one moves along at least one of the edges - call it e - incident upon v^*, and on this edge we locate a pseudo-vertex p. One such pseudo-vertex must exist, for otherwise the minimality of $\delta(v^*)$ would be contradicted. Let $r(v^*)$ be the region of $G_{\mathcal{B}}$ to which v^* belongs (known from Step 4); to locate pseudo-vertex p, we cyclically test each edge of $r(v^*)$ in turn and find the one which intersects e.(1) Clearly $\delta(p) < \delta(v^*)$.

Step 6. Locate a pseudo-vertex p adjacent to v^*, such that $\delta(p) < \delta(v^*)$. If $\delta(p) \leq 0$, then, since $\delta(v^*) > 0$, there is a point $q \in \mathcal{A} \cap \mathcal{B}$, which may be found as in Step 5; otherwise go to Step 7.

We must how consider two cases. The first is when there are no true vertices in $\mathcal{S}$ (Step 4); then the boundaries of $\mathcal{A}^*$ and $\mathcal{B}^*$ must intersect, so the point p^* may be chosen at an intersection of these boundaries and is therefore a pseudo-vertex. The second case is when $\delta(p) > 0$ (Step 6). Both these cases are treated by using an algorithm, called the wandering algorithm, which wanders among the pseudo-vertices of G^* and which uses at most $O(n)$ time. Thus we have:

(1) If v^* happens to belong to more than one region of $G_{\mathcal{B}}$, then the edges of all such regions may have to be tested to find the unique one which intersects e. In any case, the number of such tests is $O(n)$.

<u>Step 7</u>. If the test of Step 4 fails use p^*, while if the test of Step 6 fails use p, as the starting point of the wandering algorithm (to be described below), either to find a pseudo-vertex $\bar{p}$ such that $\delta(\bar{p}) \leq 0$, to which the method of Step 5 can be applied, or find a pseudo-vertex p_m such that $\delta(p_m) = \min_{v \in V^*} \delta(v)$. (We shall show below that the wandering algorithm runs in time $O(n)$.)

Before describing the wandering algorithm, we observe that the starting point of it is a pseudo-vertex, either p or p^*, which has a smaller value of δ than any true vertex. If we imagine, for purposes of proof, a contour line of δ passing through p or p^* we enclose a region $\mathfrak{R} \subseteq \mathcal{P}$ which contains a pseudo-vertex p_m having minimum $\delta(p_m)$. We note that $\mathfrak{R}$ must be convex. Also let $E'_{\mathcal{A}}$ and $E'_{\mathcal{B}}$ be the sets of edges of $G_{\mathcal{A}}$ and $G_{\mathcal{B}}$ respectively which intersect $\mathfrak{R}$. Since no true vertices lie in $\mathfrak{R}$, each edge in $E'_{\mathcal{A}} \cup E'_{\mathcal{B}}$ must separate $\mathfrak{R}$ into two convex regions. No two edges in $E'_{\mathcal{A}}$ can intersect in $\mathfrak{R}$, nor can two edges in $E'_{\mathcal{B}}$. Also, the function δ is convex downward as one travels along any edge of $E'_{\mathcal{A}} \cup E'_{\mathcal{B}}$ and its minimum must lie somewhere in $\mathfrak{R}$, because the boundary of $\mathfrak{R}$ is a contour line for δ. We shall call this point on an edge $e \in E'_{\mathcal{A}} \cup E'_{\mathcal{B}}$ where δ has a minimum value, a <u>minimum point of the edge</u> e. It is unique except in degenerate cases. The value of δ at this point will be called the <u>minimum value of the edge e</u> and denoted by min(e).

We assume momentarily that the faces of both polyhedra are triangles and, if they are not, that we shall triangulate both polyhedra. Notice that in the triangulated polyhedra the number of faces remains less than twice the number n of vertices and the number of edges remains less than three times n, so they both remain O(n). Each pseudo-vertex p' in $\mathcal{R}$ is the intersection of two edges $e_a' \in E_{\mathcal{A}}'$ and $e_b' \in E_{\mathcal{B}}'$ and is therefore shared by four regions in G*; the union of these four regions is referred to as the crown of p' and is the locus of the points which can be reached from p' without crossing any edge. Notice that e_a' is shared by two triangular faces of $G_{\mathcal{A}}$, whose union is a quadrilateral region; a similar remark holds for e_b'. Thus the crown is the intersection of these two quadrilateral regions, and the crown boundary contains either 8, or 10, or 12 pseudo-vertices (see figure 4 a, b, c, respectively). The fact that the number of crown vertices is bounded is a consequence of the hypothesis that the polyhedra have been triangulated.

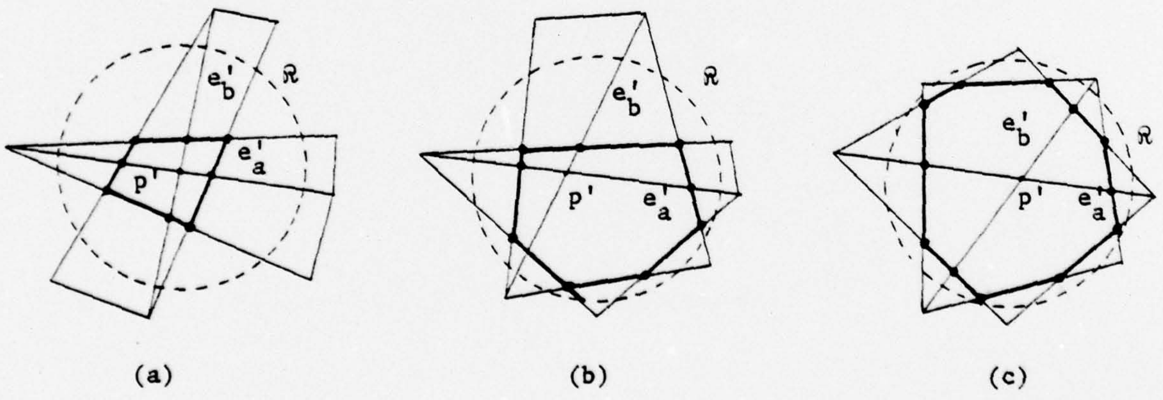


Figure 4. Illustration of the possible cases for the crown of a pseudo-vertex.

Given any pseudo-vertex p' as the intersection of e_a' and e_b', the pairs of triangles bordering these two edges can be obtained in constant time from the doubly-connected edge lists describing $\mathcal{A}$ and $\mathcal{B}$ respectively. Once these triangles are available, the pseudo-vertices in the crown can also be obtained in time bounded by a constant, and so can their values of δ. We now give the

<u>Advancing step of the wandering algorithm</u>: A pointer is moved from the current pseudo-vertex p' to a pseudo-vertex p'' which attains the minimum value of δ among all pseudo-vertices in the crown of p'.

Of course, the step is voided and the algorithm terminates if p' attains the minimum value of δ along the edges e_a' and e_b' intersecting in p': in this case if δ is positive the two polyhedra do not intersect (case (iii) in Section 5). In the other case (δ decreases either along e_a' or e_b') the advancing step is effected, and in actual practice can be carried out without exploring the entire crown of p', but simply following a path of edges along which δ decreases.

An additional algorithm simplification is that, as we shall show, polyhedra $\mathcal{A}$ and $\mathcal{B}$ need *not* be triangulated before applying the wandering algorithm. In fact, only those faces of $G_{\mathcal{A}}$ and $G_{\mathcal{B}}$ will be triangulated which are actually traversed by the wandering algorithm. Specifically, let p', the intersection of e_a' and e_b', be the current pseudo-vertex (see Figure 5). Referring for simplicity only to polyhedron $\mathcal{A}$, let f_1 and f_2 be the two faces of $G_{\mathcal{A}}$ sharing e_a'. In the doubly connected edge list of $G_{\mathcal{A}}$ we can

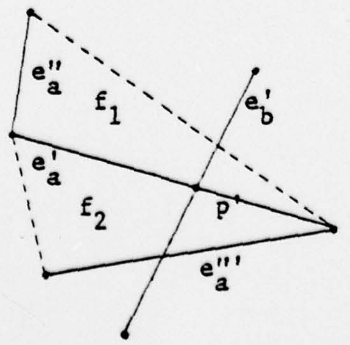


Figure 5. Partial triangulation of $\mathcal{A}$.

obtain in constant time the edges e_a'' and e_a''' which follow e_a' in the edge-sequences of f_1 and f_2, respectively. If f_1 is not a triangle, we connect the non-overlapping extremes of e_a'' and e_a', and we do likewise for f_2. The introduction of any such new edge in the doubly-connected edge list requires the modification of two pointers and the use of two other cells for construction of the appropriate record. All this can also be done in constant time. We conjecture that this insertion is not really necessary, but the present proof on the time performance of the algorithm depends upon it.

Since the wandering algorithm moves from p' to p" only if $\delta(p'') < \delta(p')$, it is obvious that the algorithm will terminate at a point p_m such that $\delta(p_m)$ is the minimum value of δ for all pseudo-vertices in $\mathcal{R}$. Even though the total number of pseudo-vertices in $\mathcal{R}$ could be $O(n^2)$, we shall now prove that the number of advancing steps is at most $O(n)$.

Recall that for an edge e in either $G_{\mathcal{A}}$ or $G_{\mathcal{B}}$, min(e) denotes the minimum value of δ on e. Let pseudo-vertex p' be the intersection of $e_a' \in E_{\mathcal{A}}'$ and $e_b' \in E_{\mathcal{B}}'$; we now define $m(p') = \max(\min(e'_a), \min(e_b'))$. Clearly $\delta(p') \geq m'(p)$.

Lemma 1: Let p' be a pseudo-vertex in $\mathcal{R}$; if $m(p') = \delta(p_m)$, then $\delta(p') = m(p') = \delta(p_m)$.

Proof: Let us assume the contrary and obtain a contradiction. In figure 6 let a' and b' represent minimum points on e_a' and e_b' respectively which are nearest to p'. By our assumption, $\delta(a') = \delta(b') = \delta(p_m)$ and hence by convexity, every point along the line segment ℓ between a' and b' also has this same δ value. Let a_1 be the pseudo-vertex closest to a' in the portion of e_a' between a' and p' (possibly, a_1 and p' coincide). The line segment ℓ crosses a region of G* bordering with $a'a_1$. Since the value of δ is linear within this region and it achieves the minimum value $\delta(p_m)$ at an interior

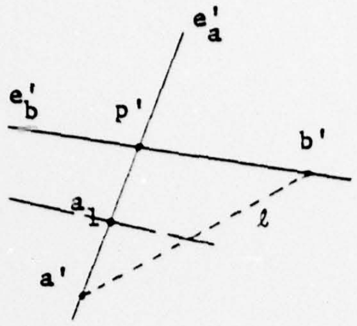


Figure 6. Illustration for the proof of $m(p') = \delta(p_m) \Rightarrow \delta(p') = m(p')$.

point, it must have this value throughout the entire region. Hence, $\delta(a_1) = \delta(p_m)$, contradicting our assumption that a' is the nearest minimum point to p' on e'_a. This proves $\delta(p') = \delta(p_m)$.□

Assuming now that $\delta(p') > \delta(p_m)$, we see by Lemma 1 that $m(p') > \delta(p_m)$. When the wandering algorithm is applied at p', it steps to a new pseudo-vertex p".

Lemma 2: $m(p'') < m(p')$.

Proof: We distinguish two cases:

(1) p' and p" do not belong to the same edge. Let p' be the intersection of e'_a and e'_b and let p" be the intersection of e''_a and e''_b (see figure 7(a)). Let p_m be the pseudo-vertex in $\mathcal{R}$ defined above.
We claim that a straight line from p_m to p" cannot intersect the interior of the region f of G* to whose boundary p' and p" belong, except at p".
In fact, if it did, any such point of intersection would, by convexity,

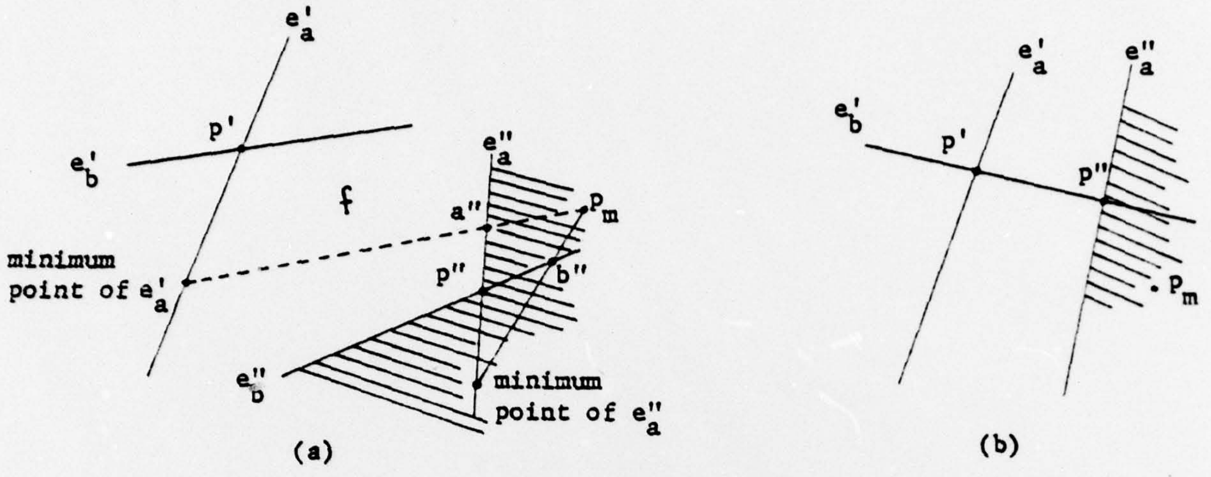


Figure 7. Illustration of the proof that $m(p'') < m(p')$.

have a value of δ as low as $\delta(p'')$. Since p'' is a minimum point of f, this would imply that all the points of f have the same value of δ, contradicting $\delta(p') > \delta(p'')$. As a consequence, either e''_a or e''_b separates p_m from p', so p_m belongs to the shaded regions in figure 7(a).

Assume, without loss of generality, that p', and hence e'_a, is separated from p_m by e''_a. Then, since e'_a does not cross e''_a in $\mathfrak{R}$, the straight line between p_m and the minimum point of e'_a intersects e''_a in a point a''. By convexity, $\min(e'_a) \geq \delta(a'')$, with equality occurring only if $\min(e'_a) = \delta(p_m)$. Assuming equality, since we have seen that $m(p') > \delta(p_m)$ and we have $\min(e'_a) = \delta(a'') = \min(e''_a) = \delta(p_m)$, we obtain $m(p') > \min(e''_a)$. Assuming instead that $\min(e'_a) > \delta(a'')$, since by definition $m(p') \geq \min(e'_a)$ and $\delta(a'') \geq \min(e''_a)$, we also obtain $m(p') > \min(e''_a)$.

Two subcases must be considered. First, assume p', and hence e'_b, is <u>also</u> separated from p_m by e''_b. Then by an identical argument $m(p') > \min(e''_b)$, so $m(p') > m(p'') = \max(\min(e''_a),\min(e''_b))$. Second, assume it is not, as shown in figure 7(a). We now show that $\min(e''_b) \leq \min(e''_a)$ thus reaching the same conclusion.

In fact, since $\delta(p'')$ is the minimum value in f, the minim point on e''_a occurs either at p'' or along e''_a on the opposite side of p'' from f. A straight line drawn between this minimum point and p_m intersects e''_b at a point b'' such that $\delta(b'') \leq \min(e''_a)$, by the convexity argument used earlier. But $\delta(b'') \geq \min(e''_b)$, whence $\min(e''_a) \geq \min(e''_b)$, as claimed.

(2) <u>p' and p'' belong to the same edge</u>. Without loss of generality, let p' be the intersection of e'_a and e'_b and let p'' be the intersection of e''_a and e'_b (see figure 7(b)). By the convexity argument, e'_a is separated from p_m by e''_a (i.e., p_m belongs to the shaded region). As in case (1), we can show that $m(p') \geq \min(e'_a) > \min(e''_a)$. To prove that $\min(e'_b) \leq \min(e''_a)$ we note that the minimum point of e''_a must be p'', for otherwise p'' would not attain the minimum of δ in the crown of p'. Thus

$m(p'') = \max(\min(e''_a), \min(e'_b)) = \min(e''_a) < m(p')$, as claimed.$\square$

<u>Theorem</u>: The number of advancing steps performed by the wandering algorithm is $O(n)$.

<u>Proof</u>: We have shown that as the wandering algorithm moves from one pseudo-vertex p' to the next, the value of $m(p')$ decreases at each step. Each value of $m(p')$ is the minimum value of one of the edges in $E'_\alpha \cup E'_\beta$. Hence, the number of distinct values which $m(p')$ can assume is no greater than $|E'_\alpha| + |E'_\beta|$, which is $O(n)$. The number of steps taken by the algorithm therefore is $O(n)$.$\square$

Since the time taken by the wandering algorithm is $O(n)$, the time taken by the entire algorithm remains $O(n\log n)$.

5. An Application: Finding a Separating Plane

The preceding method can be used to solve efficiently the important problem of linear separability in three dimensions, i.e., testing whether two finite sets of points A and B are separable by means of a plane, and, if so, finding one such plane.

Since two finite sets of points are linearly separable if and only if their convex hulls do not intersect [8], we begin by obtaining the respective convex hulls of the sets A and B by means of the Preparata-Hong algorithm [3]. Letting $|A| + |B| = n$, this task, which is completed in time $O(n\log n)$, yields two convex polyhedra $\mathcal{A}$ and $\mathcal{B}$ such that $|V_{\mathcal{A}}| + |V_{\mathcal{B}}| \leq n$. We now apply to $\mathcal{A}$ and $\mathcal{B}$ the algorithm described in Section 4: any time the algorithm declares that $\mathcal{A}$ and $\mathcal{B}$ do not intersect, we construct a separating plane.

We now recall that $\mathcal{A}$ and $\mathcal{B}$ are found to be disjoint in three exclusive cases, already referred to in Section 4:

(i) after projecting $R(\mathcal{A})$ and $R(\mathcal{B})$ on the plane (x_1, x_2), the polygons $\mathcal{A}^*$ and $\mathcal{B}^*$ are disjoint;

(ii) after evaluating δ at all true vertices of G^* we find that

$$\delta(v^*) = \min_{v \in V'_{\mathcal{A}} \cup V'_{\mathcal{B}}} \delta(v) > 0 \text{ and } v^* \text{ is an absolute minimum;}$$

(iii) after applying the wandering algorithm we find that $\delta(p_m) > 0$.

In case (i) it is sufficient to find a straight line ℓ separating $\mathcal{A}^*$ and $\mathcal{B}^*$, since a plane containing ℓ and perpendicular to the plane (x_1, x_2) separates $\mathcal{A}$ and $\mathcal{B}$. The line ℓ can be found in time $O(n)$ by an obvious modification of the Preparata-Hong algorithm for planar convex hulls ([3], p. 90).

Cases (ii) and (iii) can be handled jointly by the following considerations. Rather than constructing one separating plane, we construct a locus of separating planes and make a selection in this locus. Let u be the point at which the algorithm terminates with the result that $\mathcal{A}$ and $\mathcal{B}$ do not intersect; obviously, either $u = v^*$ or $u = p_m$. Also let u' and u" be the pre-images of u (with respect to x_3-projection) in $\mathcal{A}$ and $\mathcal{B}$, respectively. Assume at first that $u = v^*$ and, without loss of generality, let u' be a vertex (in $V_{\mathcal{A}}$). Consider the cycle F of the faces sharing u'; for each $f \in F$, imagine applying the vector $(n_1(f), n_2(f), n_3(f)) = \underline{n}(f)$ to the origin; recall that $\underline{n}(f)$ is normal to f and pointing toward the exterior of $\mathcal{A}$. Then the set of directions $\{\underline{n}(f) | f \in F\}$ defines a convex cone $C_{\mathcal{A}}$ such that any direction internal to it is normal to a supporting plane of $\mathcal{A}$. Notice now that, when $u = p_m$, point u' belongs to some edge e_a of $\mathcal{A}$ and $C_{\mathcal{A}}$ degenerates into a plane wedge delimited by the normals to the two faces of $\mathcal{A}$ which share e_a.

For u" the convex cone $C_{\mathcal{B}}$ is analogously defined, with the only modification that the directions of the vectors $\underline{n}(f)$ are reversed. The cone can assume the following forms: if $u = v^*$, then $C_{\mathcal{B}}$ is either nondegenerate, or a plane wedge, or a half-line, depending upon whether u" in $\mathcal{B}$ is either a vertex, or a point in an edge, or a point in a face, respectively; if $u = p_m$, then $C_{\mathcal{B}}$ is a plane wedge.

The solution to our problem is $C_{\mathcal{A}} \cap C_{\mathcal{B}}$. Notice, however, that this intersection consists of a single ray in the following two cases: (1) $u = p_m$, in which case they ray is the common normal to the edges which contain u' and u" in $\mathcal{A}$ and $\mathcal{B}$, respectively; (2) $u = v^*$ and u" is a point in a face of $\mathcal{B}$, in which case the ray is the normal to this face. In the remaining cases ($u = v^*$, u' is a vertex, and u" is either a vertex or a point in an edge) we first find a plane which intersects $C_{\mathcal{A}}$ in a bounded polygon; this can be done in time O(n) as follows. For each face $f \in F$, let point t(f) be the terminus of the vector $\underline{n}(t)$. Let f_1 and f_2 be two consecutive faces in the cycle F. We consider the set of planes determined by triples of points $(t(f_1), t(f_2), t(f_j))$

with $j \neq 1,2$ and $f_j \in F$. For each such plane, we call positive the half-plane bounded by the straight line ℓ through $t(f_1)$ and $t(f_2)$ and containing $t(f_j)$. The half-planes of this set have line ℓ in common and are comprised between two extreme ones, one of which intersects all the edges of C_α: the latter defines our desired plane. Next we intersect C_β with this plane and obtain either a polygon or a straight-line segment: in any case the problem is reduced to finding the intersection of two plane polygons, which can be solved in time $O(n)$ [1]. This enables us to find a vector orthogonal to a separating plane; the construction is completed by requiring that the plane contain a point internal to the segment $u'u''$.

Thus, we conclude that the construction of a separating plane of two three-dimensional sets of points, if it exists, can be effected in time $O(n\log n)$.

References

1. M. I. Shamos, Computational Geometry, Dept. of Comp. Sci., Yale University, 1977. To be published by Springer-Verlag.

2. M. I. Shamos, "Geometric Complexity," Proc. Seventh Annual ACM Symposium on Theory of Computing, pp. 224-233, May 1975.

3. F. P. Preparata and S. J. Hong, "Convex hulls of finite sets in two and three dimensions," Communications of the ACM, Vol. 20, N. 2, pp. 87-93, February 1977.

4. H. T. Kung, F. Luccio, and F. P. Preparata, "On finding the maxima of a set of vectors," Journal of the ACM, Vol. 22, No. 4, pp. 469-476, October 1975.

5. D. T. Lee and F. P. Preparata, "Location of a point in a planar subdivision and its applications," SIAM Journal on Computing, Vol. 6, N. 3, pp. 594-606, September 1977.

6. F. P. Preparata, "Location of a set of points in a planar subdivision," submitted for publication. Available in "Steps into Computational Geometry," Report ACT-1, Coord. Sci. Lab., Univ. of Illinois, Urbana, February 1977.

7. G. Ewald, Geometry: An Introduction, Wadsworth, Belmont, Calif., 1971.

8. J. Stoer and C. Witzgall, Convexity and Optimization in Finite Dimensions, I, Springer-Verlag, 1970.

Appendix

As we said in Section 2, the vertex-to-edge list of a planar graph is a collection of edge lists, referred to as <u>input edge lists</u>, stored in arrays H[1:n], VERTEX[1:2n], and NEXT[1:2n]. In the DCEL, we can identify n cycles of edges around a vertex, called <u>vertex cycles</u>, and f cycles of edges around a face, called <u>face cycles</u>. The construction of the DCEL is carried out in two phases. In the first phase, we fill the arrays V1, V2, P1, and P2, hereby constructing the vertex cycles. In the second phase we generate the names of the faces and fill the arrays F1 and F2, hereby constructing the face cycles.

Informally, phase-1 of the algorithm works as follows. The input edge lists are scanned one at a time, in the order $v_1, v_2, \ldots, v_n$. While scanning the input edge list of v_j an edge (v_j, v_i) is entered into the DCEL only if $i > j$: in this manner we ensure that each edge is entered only once. Thus any edge (v_j, v_h) with $h < j$ is already present in the DCEL, since it was entered while scanning the input edge list of v_h, earlier in the execution of the algorithm. All that is needed now is therefore the realization of the appropriate linking of such (v_j, v_h) into its position in the vertex cycle of v_j. To effect it we must determine the location of (v_j, v_h) in the DCEL. This can be done as follows,with additional storage O(n). Suppose that, while scanning the input edge list of v_h, the edge (v_h, v_j) is to be entered (obviously $h < j$). This edge is linked permanently into the vertex cycle of v_h and temporarily into a list of edges of the form (v_r, v_j), with $r < j$. The members of the latter list referred to as the <u>temporary list</u> of v_j, are linked in reverse order to that of their occurrence during the execution of the algorithm. Thus this list can be managed with only one pointer stored in an array LAST[1:n]. With these provisions, the location of (v_h, v_j) is easily obtained: in fact, prior to linking the vertex-cycle of v_j we scan the temporary list of v_j starting from LAST[j] and store the location of (v_h, v_j) into cell B[h] of an auxiliary array B[1:n]. Notice that the latter array is only scratch memory and will be used repeatedly for each v_j. Therefore the additional storage needed consists of the arrays LAST and B, both of size O(n), and of program variables a_1, a_0, u, t, r, ℓ.

We can now give the algorithm.

CONSTRUCT VERTEX CYCLES

```
1.  begin  a ← 1
2.         for j ← 1 step 1 until n do LAST[j] ← Λ (Comment: initialize LAST)
3.         for j ← 1 step 1 until n do
4.           begin    ℓ ← LAST[j]
5.                    While ℓ ≠ Λ do
6.                      begin p ← V1[ℓ]
7.                            B[p] ← ℓ
8.                            ℓ ← P2[ℓ]
                        end
                        Comment: Loop 5-8 fetches the locations of all
                        edges (v_r,v_j) with r < j by scanning the temporary
                        list of v_j and stores them into the array B.  This
                        step is obviously void for j = 1.
9.                    t ← H[j],
10.                   r ← VERTEX[t]
```

```
11.   If r > j then
12.     begin    $V1[a] \leftarrow j$,  $V2[a] \leftarrow r$
13.              $HV[j] \leftarrow a_0 \leftarrow a$, $u \leftarrow 1$
14.              $P2[a] \leftarrow LAST[r]$
15.              $LAST[r] \leftarrow a$
16.              $a \leftarrow a+1$
        end  Comment:Steps 10-15 initialize the vertex cycle for $v_j$
17.   else $HV[j] \leftarrow a_0 \leftarrow B[r]$,  $u \leftarrow 2$
      Comment: Steps 8-17 process the first member $(v_r,v_j)$
      of the input edge list of $v_j$.  If this edge was not
      previously encountered steps 11-16 are executed;
      specifically, the edge is entered in step 12.  Variable
      $a_0$ is used to denote the location of the last member
      of the vertex cycle being constructed.
18.   While  $NEXT[t] \neq H[j]$ do
19.     begin    $t \leftarrow NEXT[t]$
20.              $r \leftarrow VERTEX[t]$
21.              If r > j then
22.                begin $V1[a] \leftarrow j$, $V2[a] \leftarrow r$
23.                      $P2[a] \leftarrow LAST[r]$
24.                      $LAST[r] \leftarrow Pu[a_0] \leftarrow a$,
25.                      $a_0 \leftarrow a$,  $u \leftarrow 1$
26.                      $a \leftarrow a+1$
                   end
                 else
```

```
27.                                   begin Pu[$a_0$] ← B[r]
28.                                         $a_0$ ← B[r], u ← 2
                                      end
                          end
29.                    Pu[$a_0$] ← HV[j]
               end
end
               Comment: Steps 18-29 complete  the construction of the vertex
               cycle for $v_j$.  Specifically, loop 18-28 successively processes
               the edges incident on $v_j$ and either enters them into the vertex
               cycle (Steps 21-26) or simply links them into it (Steps 27-28).
               Step 29 closes the vertex cycle.
end
```

To evaluate the running time of the algorithm just described, notice that each edge is processed exactly twice: once to be entered into a vertex cycle and into a temporary list, the second time to be linked appropriately. Both these operations take constant time, and since the number n of edges is O(n), O(n) time is used to fill the arrays V1, V2, P1, and P2.

To complete the construction of the DCEL we must construct the face cycles. The next algorithm, CONSTRUCT FACE CYCLES, starts from the partial DCEL which is produced by the CONSTRUCT VERTEX CYCLES procedure. The algorithm will scan the DCEL, using an integer a as a counter. If F1[a] and F2[a] have already been filled,it advances to the subsequent edge; otherwise it generates the name of new face (using a counter s) and traces the edges enclosing it filling the appropriate F-fields. The algorithm terminates when 2m filling operations have been performed: an integer k is used to control this event.

CONSTRUCT FACE CYCLES

```
1.  begin  for j ← 1 step 1 until m do F1[j] ← F2[j] ← Λ
2.          a ← s ← k ← 1
3.          While k ≤ 2m do
4.            begin If F1[a] ≠ Λ and F2[a] ≠ Λ then  a ← a+1
5.                  else begin If F1[a] = Λ then u ← 1 else u ← 2
6.                             Fu[a] ← s, c ← Vu[a], HF[s] ← a_0 ← a , k ← k+1
7.                             a ← Pu[a]
8.                             While a ≠ a_0 do
9.                               begin If V1[a] = c then u ← 2 else u ← 1
10.                                    Fu[a] ← s, c ← Vu[a], k ← k+1
11.                                    a ← Pu[a]
                                 end
12.                            s ← s+1
                       end
            end
    end
```

Since in the latter algorithm each field F1[a] or F2[a] is being processed at most twice (once to be filled in steps 6 or 10, and possibly once to be just inspected in step 4), the running time is O(n). This and the analogous result for the vertex cycle algorithm substantiate our claim that the DCEL can be obtained in time O(n) from the original vertex-to-edge list.